

```

1 from keras.datasets import cifar10
2 import numpy
3
4 (x_train_ori, y_train_ori), (x_test_ori, y_test_ori) = cifar10.load_data()
5
6 print('shape of x_train: ' + str(x_train_ori.shape))
7 print('shape of y_train: ' + str(y_train_ori.shape))
8 print('shape of x_test: ' + str(x_test_ori.shape))
9 print('shape of y_test: ' + str(y_test_ori.shape))
10 print('number of classes: ' + str(numpy.max(y_train_ori) - numpy.min(y_train_ori) + 1))
11

```

↳ Using TensorFlow backend.

```

shape of x_train: (50000, 32, 32, 3)
shape of y_train: (50000, 1)
shape of x_test: (10000, 32, 32, 3)
shape of y_test: (10000, 1)
number of classes: 10

```

```

1 x_train = x_train_ori/255
2 x_test = x_test_ori/255
3 print(x_train, x_test)

1 def to_one_hot(y, num_class=10):
2     res = []
3     for ys in y:
4         code = [0]*num_class
5         code[ys[0]] = 1
6         res.append(code)
7     return numpy.asarray(res)
8
9 y_train_vec = to_one_hot(y_train_ori)
10 y_test_vec = to_one_hot(y_test_ori)
11
12 print('Shape of y_train_vec: ' + str(y_train_vec.shape))
13 print('Shape of y_test_vec: ' + str(y_test_vec.shape))
14
15 print(y_train_ori[0])
16 print(y_train_vec[0])

```

↳ Shape of y\_train\_vec: (50000, 10)  
 Shape of y\_test\_vec: (10000, 10)  
 [6]  
 [0 0 0 0 0 0 1 0 0 0]

```

1 rand_indices = numpy.random.permutation(50000)
2 train_indices = rand_indices[0:40000]
3 valid_indices = rand_indices[40000:50000]
4
5 x_val = x_train[valid_indices, :]
6 y_val = y_train_vec[valid_indices, :]
7
8 x_tr = x_train[train_indices, :]
9 y_tr = y_train_vec[train_indices, :]
10
11 print('Shape of x_tr: ' + str(x_tr.shape))
12 print('Shape of y_tr: ' + str(y_tr.shape))
13 print('Shape of x_val: ' + str(x_val.shape))
14 print('Shape of y_val: ' + str(y_val.shape))

```



```

↳ Shape of x_tr: (40000, 32, 32, 3)
   Shape of y_tr: (40000, 10)
   Shape of x_val: (10000, 32, 32, 3)
   Shape of y_val: (10000, 10)

```

```

1 batch_size = 32
2 epochs = 100
3 num_classes = 10

```

```

1 from keras import optimizers
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Activation
3 from keras.models import Sequential
4
5 model = Sequential()
6 model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
7 model.add(BatchNormalization())
8 model.add(Activation('relu'))
9 model.add(Conv2D(32, (3, 3)))
10 model.add(BatchNormalization())
11 model.add(Activation('relu'))
12 model.add(MaxPooling2D(pool_size=(2, 2)))
13 model.add(Dropout(0.25))
14
15 model.add(Conv2D(64, (3, 3), padding='same'))
16 model.add(BatchNormalization())
17 model.add(Activation('relu'))
18 model.add(Conv2D(64, (3, 3)))
19 model.add(BatchNormalization())
20 model.add(Activation('relu'))
21 model.add(MaxPooling2D(pool_size=(2, 2)))
22 model.add(Dropout(0.25))
23
24 model.add(Flatten())
25 model.add(Dense(512))
26 model.add(Activation('relu'))
27 model.add(Dropout(0.5))
28 model.add(Dense(num_classes))
29 model.add(Activation('softmax'))
30
31 opt = optimizers.rmsprop(lr=0.0001, decay=1e-6)
32 model.compile(loss='categorical_crossentropy',
33               optimizer=opt,
34               metrics=['accuracy'])
35
36 model.fit(x_train, y_train_vec,
37         batch_size=batch_size,
38         epochs=epochs,
39         shuffle=True)
40

```

```

1 score = model.evaluate(x_val, y_val, verbose=1)
2 print('Training loss: {0:.4f}\nTraining accuracy: {1:.4f}'.format(*score))

```

```

1 import matplotlib.pyplot as plt
2 %matplotlib inline
3
4 acc = history.history['acc']
5 val_acc = history.history['val_acc']
6
7 es = range(len(acc))
8
9 plt.plot(es, acc, 'bo', label='Training acc')
10 plt.plot(es, val_acc, 'r', label='Validation acc')

```



```

11 plt.xlabel('Epochs')
12 plt.ylabel('Accuracy')
13 plt.legend()
14 plt.show()

```

```

1 from keras.preprocessing.image import ImageDataGenerator
2 datagen = ImageDataGenerator(
3     featurewise_center=False,
4     samplewise_center=False,
5     featurewise_std_normalization=False,
6     samplewise_std_normalization=False,
7     zca_whitening=False,
8     zca_epsilon=1e-06,
9     rotation_range=0,
10    width_shift_range=0.1,
11    height_shift_range=0.1,
12    shear_range=0.,
13    zoom_range=0.,
14    channel_shift_range=0.,
15    fill_mode='nearest',
16    cval=0.,
17    horizontal_flip=True,
18    vertical_flip=False,
19    rescale=None,
20    preprocessing_function=None,
21    data_format=None,
22    validation_split=0.0)
23
24 datagen.fit(x_train)

```

```

1 model.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),
2                     steps_per_epoch=x_train.shape[0] // batch_size,
3                     epochs=epochs)

```

```

1 loss_and_acc = model.evaluate(x_test, y_test_vec)
2 print('loss = ' + str(loss_and_acc[0]))
3 print('accuracy = ' + str(loss_and_acc[1]))

```

## ▼ Repeat Augmentation

```

1 baseMapNum = 32
2 weight_decay = 1e-4

```

```

1 from keras import regularizers
2 from keras import optimizers
3 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Activation
4 from keras.models import Sequential
5
6 model2 = Sequential()
7 model2.add(Conv2D(baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2))
8 model2.add(Activation('relu'))
9 model2.add(BatchNormalization())
10 model2.add(Conv2D(baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2))
11 model2.add(Activation('relu'))
12 model2.add(BatchNormalization())
13 model2.add(MaxPooling2D(pool_size=(2,2)))
14 model2.add(Dropout(0.2))
15
16 model2.add(Conv2D(2*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2))
17 model2.add(Activation('relu'))

```



```

18 model2.add(BatchNormalization())
19 model2.add(Conv2D(2*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2(0.01)))
20 model2.add(Activation('relu'))
21 model2.add(BatchNormalization())
22 model2.add(MaxPooling2D(pool_size=(2,2)))
23 model2.add(Dropout(0.3))
24
25 model2.add(Conv2D(4*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2(0.01)))
26 model2.add(Activation('relu'))
27 model2.add(BatchNormalization())
28 model2.add(Conv2D(4*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2(0.01)))
29 model2.add(Activation('relu'))
30 model2.add(BatchNormalization())
31 model2.add(MaxPooling2D(pool_size=(2,2)))
32 model2.add(Dropout(0.4))
33
34 model2.add(Flatten())
35 model2.add(Dense(num_classes, activation='softmax'))
36
37 model2.summary()

```

```

1 from keras.preprocessing.image import ImageDataGenerator
2 datagen = ImageDataGenerator(
3     featurewise_center=False,
4     samplewise_center=False,
5     featurewise_std_normalization=False,
6     samplewise_std_normalization=False,
7     zca_whitening=False,
8     rotation_range=15,
9     width_shift_range=0.1,
10    height_shift_range=0.1,
11    horizontal_flip=True,
12    vertical_flip=False
13 )
14 datagen.fit(x_train)

```

```

1 batch_size = 64
2 epochs=25
3 opt_rms = optimizers.rmsprop(lr=0.001,decay=1e-6)
4 model2.compile(loss='categorical_crossentropy',
5     optimizer=opt_rms,
6     metrics=['accuracy'])
7 model2.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),steps_per_epoch=100,
8 model2.save_weights('cifar10_normal_rms_ep75.h5')
9
10 opt_rms = optimizers.rmsprop(lr=0.0005,decay=1e-6)
11 model2.compile(loss='categorical_crossentropy',
12     optimizer=opt_rms,
13     metrics=['accuracy'])
14 model2.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),steps_per_epoch=100,
15 model2.save_weights('cifar10_normal_rms_ep100.h5')
16
17 opt_rms = optimizers.rmsprop(lr=0.0003,decay=1e-6)
18 model2.compile(loss='categorical_crossentropy',
19     optimizer=opt_rms,
20     metrics=['accuracy'])
21 model2.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),steps_per_epoch=100,
22 model2.save_weights('cifar10_normal_rms_ep125.h5')
23

```

```

1 loss_and_acc = model2.evaluate(x_test, y_test_vec)
2 print('loss = ' + str(loss_and_acc[0]))
3 print('accuracy = ' + str(loss_and_acc[1]))

```



```
↳ 10000/10000 [=====] - 3s 291us/step  
loss = 0.545775131225586  
accuracy = 0.8662  
  
1 score = model2.evaluate(x_val, y_val, verbose=1)  
2 print('Training loss: {0:.4f}\nTraining accuracy: {1:.4f}'.format(*score))  
  
↳ 10000/10000 [=====] - 3s 259us/step  
Training loss: 0.4199  
Training accuracy: 0.8983
```

